MỘT SỐ KỸ THUẬT TẠO CƠ SỞ DỮ LIỆU MẪU MÃ ĐỘC

Trịnh Minh Đức*, Đinh Khánh Linh, Lê Khánh Dương, Võ Văn Trường
Trường Đại học Công nghệ thông tin & truyền thông – ĐH Thái Nguyên

TÓM TẮT

Hiện nay các cuộc tấn công bằng mã độc đang phát triển với một tốc độ rất nhanh, số lượng mã độc mới xuất hiện ngày càng nhiều, đặc biệt có nhiều loại mã độc nguy hiểm mà các phần mềm diệt virus còn chưa phát hiện được. Bên cạnh đó, những thiệt hại do mã độc gây ra ngày càng nghiêm trọng hơn. Do vậy, việc xây dựng một hệ thống có khả năng tự động nhận diện và phát hiện mã độc là hết sức cần thiết. Bài báo này trình bày ba kỹ thuật chính dựa trên dấu hiệu đó là kỹ thuật dựa trên mã băm, dựa trên mã byte và dựa trên kinh nghiệm để tạo cơ sở dữ liệu mẫu mã độc phục vụ cho quá trình quét mã độc. Các kỹ thuật này đã được thử nghiệm trên các tệp tin đã bị nhiễm mã độc trên hệ điều hành Microsoft Windows để tạo ra một bộ cơ sở dữ liệu mẫu mã độc mới. Kết quả cho thấy khi sử dụng ba công cụ quét mã độc ClamAV, Yara và Ssdeep kết hợp với cơ sở dữ liệu mẫu này, có khoảng 90% mẫu mã độc đã được phát hiện.

Từ khóa: *Mã độc máy tính; cơ sở dữ liệu mã độc; hash; byte-signature; heuristics; binary-diffing*

Ngày nhận bài: 16/4/2018; Ngày hoàn thiện: 31/8/2020; Ngày đăng: 31/8/2020

A NUMBER OF TECHNIQUES TO CREATE MALWARE DATABASE

Trinh Minh Duc*, Dinh Khanh Linh, Le Khanh Duong, Vo Van Truong
TNU - University of Information and Communication Technology

ABSTRACT

Nowadays, malware attacks are growing rapidly, the number of new malwares is appearing more and more. Many dangerous malwares can even easily bypass antivirus programs. Besides, losses caused by malwares become extremely serious. Hence, building a system which is capable of detecting malwares automatically is essential. This paper presents three main techniques based on signature to create malware database used for malware scanning process: hash, byte signature and heuristics. These techniques are applied to infected files on the Microsoft Windows operating system in order to generate a new malware database. These techniques have been tested and demonstrated the effectiveness on malware scanning tools ClamAV, Yara and Ssdeep. The results showed that when using three open source malware scan tools, including ClamAV, Yara and Ssdeep in combination with this malware sample database, about 90% of malware samples were detected.

Keywords: *Malware; malware database; hash; byte-signature; heuristics; binary-diffing*

Received: 16/4/2018; Revised: 31/8/2020; Published: 31/8/2020

* Corresponding author. *Email: duchoak15@gmail.com*

1. Mở đầu

Bài báo này giới thiệu việc tạo cơ sở dữ liệu (CSDL) mẫu mã độc cho các phần mềm chống virus. Hiện nay có rất ít thông tin liên quan đến việc tạo CSDL mẫu mã độc, chỉ có một số ít tài liệu về cách tạo CSDL với ClamAV [1]-[3] hoặc sử dụng các công cụ khác. Vì vậy, chúng tôi giới thiệu ba kỹ thuật chính để phát hiện mã độc dựa trên dấu hiệu (Signature) và tập trung chủ yếu vào tạo CSDL cho những tệp tin thực thi trên hệ điều hành Microsoft Windows. Các dấu hiệu phổ biến nhất là mã băm (Hash), mã byte (Byte-signature) và dựa vào kinh nghiệm (Heuristics). Những ý tưởng thể hiện trong bài báo có thể sử dụng để tạo CSDL mã độc trên các loại tệp khác. Ngoài ra, thông tin của bài báo có thể trợ giúp hữu ích cho các nhà nghiên cứu trong việc tạo ra các CSDL mã độc để phân loại hoặc phát hiện các tài liệu hoặc chương trình độc hại.

Cấu trúc bài báo được chia thành 5 phần: Sau phần mở đầu là phần 2, giới thiệu một số công cụ tạo CSDL mẫu mã độc; Phần 3, trình bày kỹ thuật tạo CSDL mã độc dựa trên mã băm; Phần 4, trình bày kỹ thuật tạo CSDL mã độc dựa trên mã byte; Phần 5, trình bày kỹ thuật tạo CSDL mã độc dựa trên kinh nghiệm và phần cuối cùng dành cho kết luận.

2. Một số công cụ tạo cở sở dữ liệu mẫu mã độc

Một số công cụ mã nguồn mở có thể được sử dụng để quét các tệp và được sử dụng cho các ví dụ trong bài báo này. Những công cụ này rất dễ sử dụng và có thể phù hợp với hầu hết các môi trường, chúng có thể chạy trong môi trường Linux hoặc Windows. Dưới đây là tên của công cụ và một số khả năng quét của nó. Các lệnh được trình bày kèm theo ví dụ, để người đọc có thể tự kiểm tra và phát triển. Trong bài báo này, chúng tôi đề cập đến 3 loại công cụ quét mã độc như sau:

a. Công cụ ClamAV dùng để quét mã Hexa Byte, quét biểu thức chính quy, quét MD5 và quét từng phần MD5, trong đó sigtool là công cụ để tạo các signature và mã băm [1].

b. Công cụ Yara là một công cụ quét mạnh mẽ dựa trên luật, hỗ trợ nhiều điều kiện và kiểu dữ liệu nhưng không hỗ trợ mã băm [3]-[5].

c. Công cụ Ssdeep là một công cụ để tạo và so sánh mã băm [3], [6], [7].

3. CSDL mã độc dựa trên mã băm

Kiểu dấu hiệu đơn giản và cơ bản nhất là một giá trị băm. Giá trị này được tạo ra bởi một hàm băm, là một thủ tục hoặc hàm toán học chuyển đổi một lượng lớn dữ liệu thành một giá trị đơn. Hàm băm được sử dụng phổ biến nhất là MD5 và SHA-1 [8], [9]. Các hàm băm này cực kỳ chính xác, nếu có khối dữ liệu đã được băm và sau đó cùng một khối dữ liệu như vậy có một byte thay đổi và băm lại thì các giá trị băm sẽ khác nhau. Trong ví dụ dưới đây, hai chuỗi "HelloWorld!" và "HelloWorld" chỉ khác nhau một ký tự "!" nhưng giá trị băm lại hoàn toàn khác nhau.

```
MD5 của chuỗi "HelloWorld!" =
"06e0e6637d27b2622ab52022db713ce2"
MD5 của chuỗi "HelloWorld" =
"68e109f0f40ca72a15e05cc22786f8e6"
```

CSDL mã độc dựa trên MD5 có thể được tạo ra bằng cách sử dụng công cụ ClamAV, công cụ Yara không hỗ trợ băm tệp tin và công cụ ClamAV yêu cầu hai thuộc tính để tạo dấu hiệu mã băm MD5, đó là: kích thước tệp tin theo byte và mã băm MD5. Công cụ ClamAV đi kèm với một công cụ hỗ trợ, đó là sigtool, công cụ này có thể được sử dụng để tạo dấu hiệu mã MD5. Sigtool có thể được tìm thấy trong thư mục "bin" của thư mục cài đặt ClamAV. Dưới đây là ví dụ về cách sử dụng sigtool cho tệp tin one.txt

```
C:\EXP\clamwin\bin>sigtool.exe --md5
one.txt
08bc3de154cadb33affd7d4433ebc31e:16:C
:\EXP\one.txt
```

Các mẫu dấu hiệu của ClamAV được phân cách bởi dấu hai chấm ':'. Phần thứ nhất là mã MD5

(08bc3de154cadb33affd7d4433ebc31e), phần thứ hai là kích thước và phần cuối cùng là vị trí tệp tin hoặc đầu ra. Đầu ra thường là tên mã độc hoặc một cái gì đó đặc trưng cho tệp tin. Dấu hai chấm không được sử dụng tại đầu ra bởi vì chúng được coi là các ký tự đặc biệt trong CSDL của ClamAV. CSDL mẫu MD5 cần được lưu trong một tệp có đuôi .hdb. Đầu ra từ sigtool có thể được dẫn đến (>) một tệp tin gọi là shredder.hdb. Điều này giúp dễ dàng tạo các dấu hiệu mã MD5 với tất cả các tệp trong một thư mục, sử dụng một vòng lặp đơn giản trong cửa sổ dòng lệnh (xem [1]). Ngoài ra, điều này cũng giúp tạo các dữ liệu mẫu mã độc ClamAV từ kết quả của Virustotal mà không cần có tệp tin đó, vì mã MD5 và kích thước tệp đã được cung cấp trong trình phân tích của Virustotal.

Giá trị băm rất hữu ích trong trường hợp mẫu mã độc là tĩnh, nhưng nếu một byte trong tệp thực thi bị thay đổi thì mẫu mã băm sẽ bị phá vỡ. Trong một số trường hợp mã nguồn của tệp thực thi không bao giờ thay đổi nhưng dữ liệu mà tệp thực thi thực hiện lại thay đổi lúc này cần phải sử dụng mã băm từng phần. Một ví dụ về mã nguồn không thay đổi nhưng dữ liệu thay đổi là tệp tin thực thi được tạo ra bởi bộ GUI Remote Access Trojan. Phần mã sẽ giống nhau nhưng nếu một người dùng sử dụng địa chỉ 192.168.0.1 và một người khác sử dụng 192.160.2 thì các phần dữ liệu sẽ khác nhau. Trong ví dụ dưới đây, mã băm từng phần có thể được sử dụng để nhắm tới phần mã nguồn và tạo một dấu hiệu dựa trên mã băm.

```
PESectionSize:MD5:MalwareName
.code:0970e94b5ea5bbb91b9cf963c47b489
5:MalwareName
```

Các mẫu dấu hiệu nhận biết này cần được lưu trong một tệp có đuôi .mdb để có định dạng phù hợp với ClamAV. Như đã trình bày, dấu hai chấm được coi là ký tự đặc biệt để tách dữ liệu trong CSDL mẫu của ClamAV. Phần đầu tiên là tên phần PE (.code), phần thứ hai là mã băm MD5 của phần PE này (0970e94b5ea5bbb91b9cf963c47b4895) và cuối cùng là tên của mã độc (MalwareName) (xem [1]). Mã băm cũng có thể được sử dụng để xác định các tệp tin đã bị sửa đổi hay không. Thuật ngữ Fuzzy Hashing [3], [10], [11] là một kỹ thuật mà cơ bản vẫn là nhận dạng chương trình mã độc qua mã băm, nhưng đã được bổ sung thêm các phân tích và tính toán để từ một mã băm của mã độc có thể nhận ra các mã băm "họ hàng" nhằm nâng cao khả năng phát hiện mã độc. Kỹ thuật này có thể được sử dụng để xác định các tệp tin chứa các dữ liệu đã bị xóa, sửa đổi hoặc chứa dữ liệu mới đã được chèn thêm. Dưới đây là một ví dụ so sánh hai tệp tin thực thi với sự khác biệt một byte.

```
C:\EXP>md5sum *
0bcc4aab14639664f63865cc4959fb53
second-edit.exe
1d4720491fb7d551b38ab4299a2e6787
second.exe

C:\EXP>ssdeep *
ssdeep,1,1--
blocksize:hash:hash,filename
384:cKxNLII2Fz67gMvzoH/WjCpu6JBnPLl7j
z:cKnH26xCX0,"C:\EXP\second-edit.exe"
384:BKxNLII2Fz67gMvzoH/WjCpu6JBnPLl7j
z:BKnH26xCX0,"C:\EXP\second.exe"

C:\EXP\second.exe matches
C:\EXP\second-edit.exe(99)
```

Con số ở cuối dòng trên (99) là một độ đo sự tương đồng, con số này càng lớn những tệp tin này càng tương đồng nhau.

Fuzzy hashing yêu cầu tất cả các giá trị băm được lưu trữ trong một tệp tin văn bản ASCII. Kiểu quét này đòi hỏi nhiều tính toán hơn so với các dạng quét khác do phải băm và so sánh tất cả các giá trị băm. Một vấn đề với Fuzzy hashing (cũng như với tất cả các hàm băm) là những sửa đổi dường như rất nhỏ của lập trình viên cũng có thể thay đổi đáng kể giá trị đã được tính toán. Tệp tin thực thi second.exe bên trên đã được sửa đổi bằng cách thay đổi một số chuỗi đầu ra và biên

dịch lại thành second2.exe, không đoạn mã nguồn nào bị thay đổi. Tiếp tục sử dụng công cụ ssdeep để so sánh hai tệp này như trong ví dụ dưới đây.

```
C:\EXP\ssdeep *
ssdeep,1.1--
blocksize:hash:hash,filename
384:BKxNLII2Fz67gMvzoH/WjCpu6JBnPLl7j
z:BKnH26xCX0,"C:\EXP\second.exe"
384:5KTNJIv2w05gDvzoHoRjCpP6JqnuLl7jz
:5KOCXrQQ0,"C:EXP\second2.exe"
C:\EXP\string>ssdeep
ssdeep: No input files
C:\EXP\string>ssdeep -d *
C:\EXP\string>ssdeep *
ssdeep,1.1--
blocksize:hash:hash,filename
384:BKxNLII2Fz67gMvzoH/WjCpu6JBnPLl7j
z:BKnH26xCX0,"C:\EXP\string\second.ex
e"
384:5KTNJIv2w05gDvzoHoRjCpP6JqnuLl7jz
:5KOCXrQQ0,"C:\EXP\string\second2.exe
"
C:\EXP\>ssdeep second.exe > o.txt
C:\EXP\>ssdeep second2.exe > o2.txt
C:\EXP\>ssdeep -d out*
C:\EXP\o2.txt matches C:\EXP\o.txt
(50)
```

Vì tất cả các mã băm cần được lưu trong một tệp tin văn bản sau đó được so sánh với nhau nên ssdeep có thể đòi hỏi chi phí tính toán cao [7].

4. CSDL dựa trên mã byte

Mã byte (Byte signature) là một dấu hiệu dựa trên một chuỗi các byte có mặt trong một tệp tin hoặc luồng dữ liệu. Đây là một kỹ thuật phát hiện rất phổ biến và đã được sử dụng trong chương trình quét virus đầu tiên. Tính hữu dụng của kỹ thuật này là do độ chính xác mà nó cung cấp cho việc phát hiện một chuỗi các byte. Chuỗi byte được chọn bởi vì nó tồn tại trong nhiều biến thể của mã độc từ cùng một họ. Mã byte có thể là bất kỳ loại dữ liệu nào như một đoạn mã nguồn hoặc dữ liệu được chứa bên trong một luồng dữ liệu hoặc một tệp thực thi, tệp PDF (Portable Document File) hoặc tài liệu Word. Bên dưới là một đoạn mã byte trong định dạng của ClamAV cho tệp second.exe.

```
Seconddotexe:1:77CF2463CC0263AB0185E6
```

Trong đó "Seconddotexe" là đầu ra được hiển thị bởi chương trình quét ClamAV; đây thường là tên của một họ mã độc. Phần thứ hai là dành cho chương trình quét của ClamAV biết được loại tệp tin của tệp tin đã được quét. Giá trị '1' để chương trình quét nhận biết các tệp tin thực thi di động (PE files). Để quét bất kỳ loại tệp nào, cần phải sử dụng giá trị '0'. "77CF2463CC0263AB0185E6" là biểu diễn thập lục phân của đoạn chương trình hợp ngữ dưới đây.

```
start: 0x302C2F  length: 0xC
77     nop
CF 24     call    dword ptr [esi]
63 CC 02    add     esi, 4
63 AB 01    add     ebx, 1
85 E6     jnz     short loc_302F30
```

Trong định dạng của Yara đoạn mã byte được biểu diễn như sau.

```
rule example
{
 strings:
 signature = { 43 77 CF 24 63 CC 02
 63 AB 01 85 E6 }
 condition:
 signature
}
```

Định dạng dấu hiệu nhận dạng mã độc của Yara khác nhiều so với ClamAV. Phong cách cú pháp tương tự như một cấu trúc của ngôn ngữ C. Chuỗi đầu tiên xác định rằng chúng ta đang tạo ra một luật của "example". Các "strings" và "condition" là các từ khóa được sử dụng bởi chương trình quét Yara. Các “strings” sẽ xác định các dấu hiệu nhận biết

chương trình mã độc. Dấu hiệu nhận biết của Yara có thể là bất kỳ định dạng nào từ chuỗi (strings), hex-byte, regex và một số định dạng khác. Từ khóa "condition" xác định trong những trường hợp nào Yara sẽ cảnh báo về dấu hiệu nhận biết. Có rất nhiều điều kiện có thể xây dựng cho Yara để phát hiện một tệp tin (xem [2]).

Binary diffing

Các kỹ thuật phân tích thủ công và tự động có thể được sử dụng để tìm các khối mã có mặt trong các biến thể. Quá trình so sánh nhiều tệp tin thực thi cho các điểm tương đồng được gọi là Binary diffing [3], [12], [13]. Binary diffing theo cách thủ công bao gồm việc xem xét mã hợp ngữ của nhiều tệp tin và ghi lại các phần mã tồn tại trong các tệp tin. Khi các khối mã được xác định, một sự so sánh liên tiếp có thể được thực hiện để kiểm tra sự tương đồng. Nếu mã có độ tương đồng cao, nó có thể là một ứng cử viên tốt cho cách nhận biết byte-signature. Một cách tiếp cận bán tự động của kỹ thuật Binary diffing bao gồm việc nhóm các tệp tin thành các tệp tin tương đồng nhau, sử dụng một công cụ để dịch mã máy thành mã hợp ngữ (disassembler) và ghi vào một tệp văn bản, sau đó so sánh và phân tích các đoạn mã hợp ngữ tương đồng này. Chúng ta có thể sử dụng công cụ IDA (Interactive Disassembler) [3], [14], [15] để tạo mã hợp ngữ và sau đó sử dụng KDiff (công cụ mã nguồn mở để so sánh sự khác nhau của các tệp tin) [16] như ví dụ sau.

```
C:\EXP\>dir
     23,503 second.exe
     23,503 second2.exe
C:\EXP\>"C:\ProgramFiles\IDAFree\idag
.exe" -B "second.exe"
C:\EXP\>"C:\ProgramFiles\IDAFree\idag
.exe" -B "second2.exe"
C:\EXP\>dir
     51,262 second.asm
     23,503 second.exe
    204,956 second.idb
     51,274 second2.asm
     23,503 second2.exe
    204,956 second2.idb
```

Cờ "-B" là để tạo ra một CSDL cho IDA (.idb) và một đầu ra là tệp mã hợp ngữ (.asm). KDiff là một công cụ tốt để so sánh các tệp tin bởi vì nó không thực hiện một sự so sánh trực tiếp giữa hai tệp tin, nó sẽ thử sắp xếp các đoạn mã bị tách biệt bởi một lượng lớn dữ liệu hoặc mã. Các phần này có thể khác nhau do một chức năng có thể không nằm trong một tệp thực thi hoặc có sự xuất hiện mã hoặc dữ liệu rác. KDiff cũng có thể so sánh đến ba tệp tin cùng một lúc. Các công cụ tự động như Bindiff [3], [17], PatchDiff2 [3] và DarunGrim [3] có thể được sử dụng để tìm kiếm sự khác biệt giữa các tệp CSDL IDA. VxClass [3] là một công cụ khác có thể được sử dụng để tạo ra các bộ biến thể, so sánh và xác định các phần mã có thể được sử dụng cho sự phát hiện dựa trên dấu hiệu. Quá trình so sánh cũng rất hữu ích cho việc hiển thị các đoạn mã nơi các ký tự đại diện cần phải được sử dụng cho sự phát hiện dựa vào byte-signature. Các ký tự đại diện sẽ cho phép chương trình quét bỏ qua các byte mã không cố định trong nhiều biến thể. Các byte không cố định này có thể bị gây ra bởi việc chèn mã rác, bổ sung hoặc loại bỏ mã hay dữ liệu. Một sự thay đổi nhỏ trong bố cục tệp tin có thể phá vỡ quá trình phát hiện dựa vào byte-signature. Các lệnh phổ biến sử dụng độ lệch (offset) dựa trên bố cục tệp là long jmps, lời gọi các hàm con (sub-routine), các API hoặc các lệnh khác như cmp, mov, sub, add… Cách tốt nhất là khi tạo các signature, ta luôn thêm các ký tự đại diện (wildcard) cho các offset, mặc dù chúng vẫn giữ nguyên trong nhiều biến thể. Ví dụ dưới đây cho thấy bố cục tệp tin có thể phá vỡ quá trình phát hiện dựa vào byte-signature. Hai khối mã trong hai tệp three.exe và three2.exe, địa chỉ bắt đầu, chức năng và chiều dài đều giống hệt nhau. Sự khác biệt duy nhất giữa hai khối mã là độ lệch địa chỉ của dword được so sánh với ebx. Sự khác nhau trong bố cục tệp tin là do các chuỗi đầu ra có độ dài khác nhau.

Khối mã trong three.exe

```
start: 0x412368 length: 0x14
62 EC 6B 23 52 00 cmp ebx, offset
dword_52236B
1E 63 2B FF FF FF jnb loc_411236
CB 22 EC 23 00 mov esi, 23EC22h
6E 8B F2 lea edi, [ebp+var_20]
62 EC 6B 23 52 00 1E 63 2B FF FF FF
CB 22 EC 23 00 6E 8B F2
```

Khối mã trong three2.exe

```
start: 0x412368 length: 0x14
62 EC 73 23 52 00 cmp ebx, offset
dword_522373
1E 63 2B FF FF FF jnb loc_411236
CB 22 EC 23 00 mov esi, 23EC22h
6E 8B F2 lea edi, [ebp+var_20]
62 EC 73 23 52 00 1E 63 2B FF FF FF
CB 22 EC 23 00 6E 8B F2
```

Trong định dạng của ClamAV chúng ta có thể biểu diễn cho cả hai tệp này như sau.

```
Both three dot exe
:1:62EC********1E632BFFFFFFCB22EC2300
6E8BF2
```

Trong định dạng của Yara chúng ta có thể biểu diễn như sau.

```
rule both_three_dot_exe
{
strings:
signature = { 62 EC ?? ?? ?? ?? 1E 63
2B FF FF FF CB 22 EC 23 00 6E 8B F2 }
condition:
signature
}
```

5. CSDL dựa vào kinh nghiệm

Kỹ thuật phát hiện cuối cùng là dựa vào kinh nghiệm (heuristics). Heuristics được sử dụng khi chương trình mã độc hại quá phức tạp đối với mã băm và byte-signature. Heuristics là thuật ngữ chung cho các kỹ thuật khác nhau được sử dụng để phát hiện chương trình độc hại dựa vào hành vi của chúng. Đây là một trong những dạng phát hiện phức tạp nhất. Một công cụ chống virus có thể sử dụng các kỹ thuật phân tích như giả lập môi trường, API hooking, sand-boxing, các bất thường trong tệp tin… để phát hiện mã độc. Mỗi công cụ chống virus sử dụng các thuật toán khác nhau và kỹ thuật độc quyền khác nhau. Một ví dụ đơn giản về việc tạo ra một dấu hiệu nhận biết dựa trên heuristics gồm có một API logger và các luật (rules) dựa trên các API. Hãy bắt đầu với chuỗi "Hello World" của mã độc Poison Ivy (một loại Trojan trên hệ điều hành Microsoft Windows). Chúng ta có thể sử dụng một máy chủ làm ví dụ để tạo một dấu hiệu nhận biết dựa trên luật API (API rule). Khi Poison Ivy được thực thi nó sẽ tạo ra một mutex, thêm một khóa (key) vào registry và sao chép chính nó vào thư mục System32. Mutex mặc định cho Poison Ivy là ")! VoqA.I4". Bằng cách sử dụng chương trình giám sát Kerberos API Monitor chúng ta có thể mô phỏng một API hook (có thể được sử dụng bởi một công cụ chống virus). Một luật API dựa trên heuristics cho Poison Ivy có thể như sau.

```
Rule A
An API call to RtlMoveMemory with a
string of
"SOFTWARE\Classes\http\shell\open\com
mandV"
Rule B
An API call to CreateMutexA with a
string of ")!VoqA.I4"
Rule C
An API call to GetSystemDirectory
if ( Rule A then Rule B then Rule C )
then
Process  = PoisonIvy
Keribos Output
Rule A
second.exe   | 00401447 |
RtlMoveMemory(0012F458, 0040162F:
"SOFTWARE\Classes\http\shell\open\com
mandV", 00000028) returns: 0012F458
...........
```

```
Rule B
second.exe    | 0040155D |
CreateMutexA(00000000, 00000000,
0012F43B: ")!VoqA.I4") returns:
0000003C
..........
Rule C
second.exe    | 004018BF |
GetSystemDirectoryA(0012F6F1,
000000FF) returns: 00000013
```

6. Kết luận

Bài báo trình bày một số kỹ thuật để tạo CSDL mẫu mã độc cho các chương trình chống virus. Chúng tôi hy vọng các kết quả trình bày trong bài báo này sẽ hữu ích và có thể cung cấp một số ý tưởng cho những người làm về lĩnh vực phân tích phần mã độc. Trong thời gian tới chúng tôi sẽ tiếp tục nghiên cứu mở rộng kỹ thuật phát hiện dựa trên kinh nghiệm từ đó kết hợp với các thuật toán học máy vào quá trình xây dựng các hệ thống có khả năng tự động nhận diện các dạng mã độc một cách chính xác.

Lời cảm ơn

Bài báo là kết quả nghiên cứu của đề tài cấp Đại học Thái Nguyên, tên đề tài: "Xây dựng phần mềm diệt virus ictuav", mã số đề tài ĐH2016-TN07-01. Chúng tôi xin gửi lời cám ơn đến các thành viên trong nhóm đã nhiệt tình giúp đỡ trong quá trình nghiên cứu.

TÀI LIỆU THAM KHẢO/ REFERENCES

[1]. T. Kojm, *Clamav User manual*, Cisco Systems, Inc, 2016.

[2]. K. D. Nguyen, T. H. Tran, and N. T. Tran, "Memory-Based Multi-pattern Signature Scanning for ClamAV Antivirus," *In Proc. FDSE 2014 Springer Cham*, vol. 8860, pp. 58-70, 2014.

[3]. M. H. Ligh, S. Adair, B. Hartstein, and M. Richard, *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*, Wiley Publishing, Inc, 2011.

[4]. R. Dias, "Intelligence-Driven Incident Response with YARA," *The SANS Institute*, 2014. [Online] Available: https://www.sans.org/reading-room/whitepapers/forensics/intelligence-driven-incident-response-yara-35542 [Accessed Dec. 2, 2018].

[5]. V. M. Álvarez, *YARA User's Manual*, 2014.

[6]. J. Kornblum, "Ssdeep'documentation," 2015. [Online] Available: https://ssdeep-project.github.io/ssdeep/. [Accessed Dec. 2, 2018].

[7]. J. Kornblum, "Identifying Almost Identical Files Using Context Triggered Piecewise Hashing," *Digital investigation 3S, Elsevier*, pp. 91-97, 2006.

[8]. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied Cryptography*, 1996.

[9]. W. Stallings, *Cryptography and Network Security Principles and Practices, Fourth Edition*, Prentice Hall, 2005.

[10]. Y. Li, S. C. Sundaramurthy, A. G. Bardas, X. Ou, D. Caragea, X Hu, and J. Jang, "Experimental study of fuzzy hashing in malware clustering analysis," *In Proc. CSET'15 Proceedings of the 8th USENIX Conference on Cyber Security Experimentation and Test*, August 2015.

[11]. J. D. Dodson, and A. Siraj, "Applying Fuzzy Hashing to Steganography," *International Journal of Future Computer and Communication*, vol. 4, no. 6, pp. 421-425, 2015.

[12]. J. Ming, and D. Xu, "BinSim:Trace-based Semantic Binary Diffing via System Call Sliced Segment Equivalence Checking," In Proc. 26th USENIX Security Symposium, 2017.

[13]. Z. Wang, K. Pierce, and S. McFarling, "BMAT - A Binary Matching Tool for Stale Profile Propagation," *The Journal of Instruction-Level Parallelism (JILP)*, vol. 2, no. 2, pp. 228-247, May 2000.

[14]. V. Pirogov, *Disassembling Code: IDA Pro and SoftICE*, A-LIST Publishing, 2006.

[15]. J. Ferguson, D. Kaminsky, J. Larsen, L. Miras, and W. Pearce, *Reverse Engineering Code with IDA Pro*, Andrew Williams, USA, 2008.

[16]. J. Eibl, "The KDiff3 Handbook," The Free Software Foundation, 2007. [Online] Available: http://kdiff3.sourceforge.net/doc/index.html [Accessed Dec. 1, 2018].

[17]. Google LLC, *BinDiff Manual Version 4.3*, 2017. [Online] Available: https://www.zynamics.com/bindiff/manual/ [Accessed Dec. 2, 2018].